Autonomous Racing Controller Using Deep Reinforcement Learning in TORCS

Arshman Khawar 22I-2427 Rehan Tariq 22I-0965

May 11, 2025

Abstract

This report details the development of an autonomous racing controller for the TORCS simulator, utilizing a Deep Deterministic Policy Gradient (DDPG) algorithm to navigate complex tracks and avoid collisions. The controller leverages a custom environment interface, enhanced state preprocessing, and a tailored reward function to optimize speed, track adherence, and recovery from adverse situations. Despite challenges with initial instability and recovery behaviors, the implemented system demonstrates competitive performance, achieving smooth navigation and robust obstacle avoidance across varied track conditions.

1 Introduction

The Open Racing Car Simulator (TORCS) provides a dynamic platform for testing autonomous driving algorithms, simulating real-world racing scenarios with rich telemetry data. This project aims to create a controller capable of racing against opponents while optimizing speed, maintaining track position, and avoiding collisions. Using the DDPG algorithm, we address the continuous action space of steering, acceleration, and braking, enabling the agent to learn sophisticated driving strategies through interaction with the environment. The controller operates within a Python-based client, interfacing with the TORCS server via UDP, and adheres to the project's requirement for a non-rule-based approach.

2 Methodology

The methodology integrates the DDPG algorithm with a custom TORCS environment, focusing on state preprocessing, action execution, and reward design to achieve robust racing performance.

2.1 Deep Deterministic Policy Gradient (DDPG)

DDPG, an actor-critic method, is well-suited for continuous control tasks like autonomous racing. The actor network outputs actions (steering, acceleration, braking), while the critic evaluates their expected rewards. Both networks use TensorFlow with Keras for implementation, featuring batch normalization to stabilize training.

2.2 Neural Network Design

The actor network processes a 30-dimensional state vector, including normalized track sensors, speed, angle, and a derived track curvature feature. It comprises two hidden layers (64 and 32 units, ReLU activation) and outputs three actions: steering (tanh, [-1, 1]), acceleration (sigmoid, [0, 1]), and braking (sigmoid, [0, 1]). The critic network takes state and action inputs, with similar hidden layers, outputting a single Q-value. The architecture ensures efficient learning of complex driving policies.

2.3 Environment Setup

The TORCS environment is wrapped using a custom Gym interface (`torcs_env.py`), processing raw telemetry data (track sensors, speed, RPM, opponent distances) into a normalized state. Key features include:

- **Track Curvature**: Standard deviation of track sensor differences, aiding turn anticipation.
- **Opponent Proximity**: Normalized distances to nearby cars, enhancing collision avoidance.
- **Normalization**: Speed divided by 300, angles by π, track distances by 200, ensuring stable inputs.

The environment supports reverse gear for recovery, activated when speed is low (< 5 km/h) and the car is off-track or near opponents.

2.4 Reward Function

The reward function is designed to balance speed, track adherence, and safety:

- **Progress**: $speedX \cdot \cos(angle)$, rewarding forward motion aligned with the track.
- **Track Penalties**: -20 for off-track ($track.min() < 0$), $-5 \cdot |trackPos|$ for straying ($|trackPos| > 1$).
- **Collision Penalty**: -10 if opponent distance < 10 units.
- **Alignment Bonus**: $+5 \cdot \cos(angle)$ for track alignment ($\cos(angle) > 0.9$).
- **Recovery Bonus**: +10 for reversing ($gear = -1$, $speedX < 0$) when stuck.

This structure incentivizes fast, centered driving while promoting recovery from adverse situations.

2.5 Training Process

Training uses a replay buffer (capacity 100,000) to store experiences, sampled in batches of 32. Ornstein-Uhlenbeck noise (steering: $\theta = 0.4$, acceleration: $\theta = 0.03$, braking: $\theta = 0.02$) facilitates exploration. Key hyperparameters include:

- Actor learning rate: 0.0003
- Critic learning rate: 0.002
- Soft target update: $\tau = 0.001$
- Discount factor: $\gamma = 0.99$

Action smoothing ($\alpha = 0.5$) reduces jitter, and recovery logic overrides actions when stuck, setting $accel = 0$, $brake = 0.5$, and random steering.

3 Implementation Details

The project comprises three main files:

- `ddpg.py`: Implements the DDPG algorithm, including actor-critic networks, replay buffer, and training loop. It handles state preprocessing, action execution, and model saving.
- `race_environment.py`: Defines the Gym environment, managing TORCS server communication, state processing, and reward calculation. It includes reverse gear logic and track curvature.
- `Main_Driver.py`: Facilitates low-level UDP communication with the TORCS server, unchanged from the provided template.

The system was tested on a standard track configuration, with rendering enabled for visual debugging. Challenges included initial instability (addressed via batch normalization) and poor recovery (improved with reverse gear and reward shaping).

4 Results

The DDPG controller achieves competitive performance, completing laps with minimal collisions. Key observations:

- **Speed**: Maintains stable speeds (50–150 km/h) on straightaways, with smooth gear transitions (1–6, RPM-based).
- **Track Adherence**: Keeps *trackPos* near 0, with occasional off-track events mitigated by reverse gear.
- **Collision Avoidance**: Avoids opponents effectively (opponent distance typically > 10 units), though sharp turns remain challenging.
- **Recovery**: Reverses successfully when stuck, reducing early terminations (episodes last 50–100 steps).

Training over 2000 episodes shows increasing total rewards, stabilizing around 200–300 per episode, indicating robust learning.

5 Discussion

The controller excels in straight-line driving and basic collision avoidance but struggles with sharp turns, where delayed steering leads to collisions. The reverse gear logic significantly improves recovery, though fine-tuning is needed for seamless integration with the learned policy. Future enhancements could include:

- Incorporating vision-based inputs for richer state information.
- Adding multi-agent training to better handle opponent interactions.
- Exploring Proximal Policy Optimization (PPO) for potentially stabler training.

The modular design allows easy experimentation with different algorithms or reward structures.

6 Conclusion

This project successfully implements a DDPG-based controller for TORCS, achieving smooth, fast, and safe racing performance. By addressing challenges like recovery and responsiveness through custom reward design and environment enhancements, the controller demonstrates the potential of deep reinforcement learning in autonomous racing. The framework provides a scalable foundation for future research in self-driving systems.